Exploring on-node parallelism with `neutral`, a Monte Carlo neutral particle transport mini-app

Matt Martineau
HPC Group
University of Bristol
Bristol, United Kingdom
m.martineau@bristol.ac.uk

Simon McIntosh-Smith
HPC Group
University of Bristol
Bristol, United Kingdom
cssnmis@bristol.ac.uk

***Abstract*—In this research we describe the development and optimisation of a new Monte Carlo neutral particle transport mini-app, `neutral`. In spite of the success of previous research efforts to load balance the algorithm at scale, it is not clear how to take advantage of the diverse architectures being installed in the newest supercomputers. We explore different algorithmic approaches, and perform extensive investigations into the performance of the application on modern hardware including Intel Xeon and Xeon Phi CPUs, POWER8 CPUs, and NVIDIA GPUs.**

When applied to particle transport the Monte Carlo method is not embarrassingly parallel, as might be expected, due to dependencies on the computational mesh that expose random memory access patterns. The algorithm requires the use of atomic operations, and exhibits load imbalance at the node-level due to the random branching of particle histories. The algorithmic characteristics make it challenging to exploit the high memory bandwidth and FLOPS of modern HPC architectures.

Both of the parallelisation schemes discussed in this paper are dominated latency issues caused by poor data locality, and are restricted by the use of atomic operations for tallying calculations. We saw a significant improvement in performance through the use of hyperthreading on all CPUs and best performance on the NVIDIA P100 GPU. A key observation is that architectures that are tolerant to latencies may be able to hide the negative characteristics of the algorithms.

I. Introduction

The DoE is preparing for their next-generation supercomputers: Trinity, Sierra and Summit. Trinity will be comprised of more than 19,000 nodes, half of which will be Intel Xeon CPUs, and the other half Intel Xeon Phi processors codenamed Knights Landing (KNL) [1]. Sierra and Summit will consist of between 3,000 and 4,000 nodes, each containing dual socketed POWER 9 CPUs, and several NVIDIA Volta GPUs [2], [3].

The implication of this architectural diversity is that a number of critical applications will need major alterations to take advantage of the resources. Many of those applications will already have highly optimised MPI implementations, and some will use OpenMP to take advantage of node-level threading. We have observed cases where achieving on-node performance is more challenging when targeting Intel Xeon Phi processors and NVIDIA GPUs [4], [5], [6].

A significant amount of research is being conducted into the improvements required at the programming environment level, and application level, to exploit the modern architecture stack. This paper aims to contribute to this effort through an investigation of the performance of Monte Carlo neutral particle transport.

Neutral particle transport plays a major role in simulations for domains such as astrophysics, and medical and nuclear sciences, and is commonly solved using either deterministic or stochastic methods [7], [8]. In this paper we present a new Monte Carlo neutral particle transport mini-app, `neutral`[1], and discuss how we optimised it for modern processors.

There are several challenges involved in parallelising Monte Carlo neutral particle transport, including load-balancing, optimal threading strategies, and managing cross-sectional lookups. While we will acknowledge each of these issues, there is already a wealth of research that focusses upon the load balancing in the context of Monte Carlo particle transport, for example [9]. The primary focus of this paper is achieving the optimal on-node performance on modern architectures.

II. Contributions

In this paper we present several contributions:

- A new mini-app, `neutral`, which is a small, but representative, Monte Carlo neutral particle transport solver.
- Analysis of the limiting features of the application on modern HPC hardware: Intel Xeon and Xeon Phi CPUs, POWER8 CPUs, and NVIDIA GPUs.
- Suggestions for optimisations and algorithmic approaches that should be used for best performance when parallelising a Monte Carlo particle transport application.

III. Background

The Monte Carlo method can be used to solve a number of important classes of algorithms. In many cases, the approach is considered to be embarrassingly parallel, as the workloads can be partitioned without any explicit dependencies [10]. When applied to neutral particle transport, the Monte Carlo approach tracks individual particles, which are independent in principal, but those particles interact with a mesh, introducing a dependency. In our mini-app, tallies are stored of the energy deposited by particles moving through the mesh, and it is this mesh dependence that represents one of the most significant challenges in efficiently parallelising the algorithm on modern architectures.

[1] https://github.com/uob-hpc/neutral

A. Transport

Particle transport algorithms seek to trace the movement of particle densities through some geometry. As those particles move, they interact with the materials they pass through, potentially depositing energy. In reactor simulations, particle transport is essential for shielding and criticality calculations, while for medical sciences the algorithms can be used to determine radiation dosages [11], [8].

Particle transport calculations generally require solving some form of the Boltzmann particle transport equation. The deterministic approach implicitly solves the equation through considering the average behaviour of particles in a discretisation of the problem. The Monte Carlo method statistically determines the solution by considering the individual physical phenomena occurring during a large number of particle histories, and can be considered a numerical integration of the transport equation and tally response function.

The core method relies heavily upon the central limit theorem, expecting that simulating enough particles will lead to an accurate observation of the mean behaviour of those particles, converging upon a close approximation to the real solution. Readers interested in additional information about the Monte Carlo and deterministic transport methods can refer to the sources used in the development of the mini-app [7], [12].

IV. Neutral Mini-App Design

The `neutral` mini-app is a member of the `arch` project[2] developed at the University of Bristol, which is a suite of mini-apps that act as proxies for key application classes. The mini-app is intended to represent the performance profile of larger applications, and the initial design is such that important performance bottlenecks such as cross-sectional lookups and mesh tallying are treated in a representative manner.

A. Particle Event Tracking

The particle event tracking procedure for Monte Carlo neutral particle transport considers several major events, that are depicted in Figure 1.

- *collision events* - As particles move through space, they will interact with the nuclei of the material that they are being transported through. There are several interactions that can occur, of which we consider absorption, and elastic scattering.
- *facets events* - The particles move through continuous space, but are dependent upon the material properties stored at their position in the decomposed computational mesh. As particles reach the facets of their containing mesh cell, a tally of any deposited energy must be stored, and the density of the destination cell can be loaded. During multi-node execution, a facet encounter can result in the particle moving between processes.
- *census events* - The terminal event that occurs once the particle has reached the end of the timestep.

[2]https://github.com/uob-hpc/arch

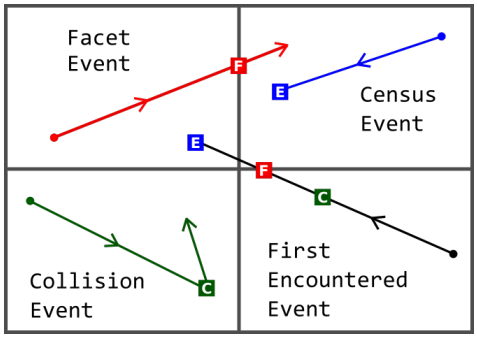

Fig. 1. Four single-event particle tracks are shown inside the computational mesh. The First Encountered Event for the particle in the bottom right is the *collision* event.

In order to determine which events will be encountered by the particle along its track, it is necessary to maintain individual timers for each event. Each time an event is encountered and handled, the timers for the other events must be updated accordingly to account for the distance travelled by the particle. While testing the mini-app, the time to *census* was set to a small fixed size in order to control the number of events that occurred per timestep.

We consider two types of *collision* event: elastic scattering, and absorption. Elastic scattering occurs when a neutral particle collides with a nucleus and scatters in some direction, conserving kinetic energy. The absorption event is where the collision leads to the particle, and therefore it's energy, being completely absorbed by the encountered nucleus.

B. Test Problems

We will later demonstrate that many aspects of the performance profile are problem dependent, and so the selection of appropriate test problems is critical to uncovering the key characteristics of this particular application. Throughout the paper we will present results for the mini-app using three distinct test cases (Figure 2). Each has been chosen to expose the limiting behaviour, or represent a realistic problem setup.

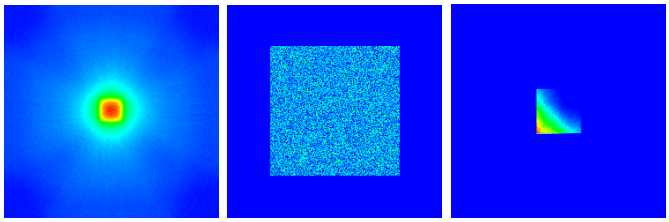

Fig. 2. Plots of the energy deposition for the three test problems used in this research, after a single timestep.

- The *stream* problem (left), where particles start in the center of the space and move rapidly across a mesh of homogeneously low mass density ($1.0 \times 10^{-30} Kg/m^3$). Due to reflective boundaries, a particle may travel multiple times across the whole width of the mesh. Around 7000 facets are encountered per simulated particle.

- The *scatter* problem (middle), where the mass density of the mesh is homogeneously high ($1.0 \times 10^3 Kg/m^3$). Many of the particles will not leave the cell that they were born in, rather they will deposit energy until their energy falls below the fixed value of interest.
- The *center square problem, csp,* (right), where particles start in the bottom left of the mesh and stream across the space. The mesh is low density everywhere except a square in the center of the space, which is high density.

In all test cases we have chosen a timestep of $10^{-7}s$, and mesh dimensions of 4000^2. The timestep is set in order to make runtimes acceptable, and different domains will have different timestep requirements, while the mesh dimensions have been chosen to represent a large but realistic test problem. For the *stream* and *csp* test cases we simulate 10^6 particles, and for the *scatter* test we simulate 10^7 particles.

Throughout the paper we will generally present the results of all test cases; however, the *csp* problem is the most realistic, making the results more relevant to real computational workloads. We expect that the *scatter* and *stream* problems will instead assist in discovering the root causes of performance differences on the target architectures.

C. Intersection Checking

To perform *facet* intersection checking, we are able to leverage the simple geometry of the mesh to calculate the new direction. The problem is essentially solved as a simple intersection in Cartesian space.

One of the key benefits to using the Monte Carlo approach, when compared to deterministic transport, is that the algorithm is able to handle complex geometric constructions [13]. While this is an important feature from a scientific perspective, we hypothesise that it is less important from a computational perspective. In our application we have chosen to use a two-dimensional structured grid in order to expose those issues that are independent of the geometry. We will extend the application in the future to support three-dimensional unstructured geometry, to validate our current assumptions.

We currently enforce reflective boundary conditions, increasing the particle lifetime and making it straightforward to track the conservation of the particle population.

D. Cross-Sectional Data

In order to determine if a *collision* event has occurred, we have to perform a lookup of cross sectional data. We currently consider a homogeneous non-multiplying media, and as such do not have to deal with the emission of secondary particles through any processes such as fission. Two dummy data tables have been generated that mimic the capture and scatter cross sections for a single material, each of which is loaded into memory at the beginning of a given simulation.

We have attempted to make the size of the tables representative of the nuclear data lookup tables that might be used in a real application. It is well known that the lookup tables can be large and cause a bottleneck in real applications [14]. Although not the focus of this research, in the future we plan to investigate the impact of managing multiple diverse materials in a single simulation.

There are two forms of neutron cross section that are needed in `neutral`:

1) Microscopic Cross Sections: To calculate the time until the next *collision* event, a search is performed to find the energy bin for the particle's continuous energy, and a linear interpolation gives an accurate approximation to the true microscopic cross section.

2) Macroscopic Cross Sections: Are calculated by scaling the microscopic cross sections by the mass density of the cell that a particle resides within. As such, a dependency is introduced between each particle and the computational mesh containing cell centered densities.

E. Variance Reduction Techniques

There are a number of variance reduction techniques discussed in the Monte Carlo particle transport literature, and we implement several of them to improve the overall accuracy of our results [7], [15].

In a typical analogue calculation, particles stream until they are absorbed, where their contribution is immediately tallied and their life ends. We instead artificially extend the lifetime of the particles by giving them each a weight, allowing each particle to represent a group of particles. Upon absorption, the individual particle weights are reduced, rather than declaring them as dead particles. Only once the weight has reduced past a fixed point, or the particle has reached a low enough energy, do we terminate the particle history.

F. Random Number Generation

Random number generation is the foundation of Monte Carlo based applications. When selecting a random number generator (RNG), there are a number of considerations, including parallelisability, statistical robustness, reproducibility and period. While we do not intend for the mini-app to solve real scientific problems, it was important to select a random number generator that would be suitable for use in real applications. This would ensure that we could accurately factor in the cost of random number generation on diverse architectures.

We decided to use Random123, which is a suite of counter-based RNGs (CBRNGs), in particular using the Threefry method [16]. CBRNGs are stateless and deterministically respond with a random number when given a key-counter pair. This allows us to store a key-value pair per particle, and achieve reproducibility between runs for the purpose of testing during debugging. Also, Random123 provides interfaces that are suitable for parallel generation with OpenMP and CUDA, which met our requirements for all architectures.

In `neutral`, random numbers determine the initial particle locations and directions within a bounded source region. Also, when a *collision* event occurs and a particle scatters, random numbers are generated to determine the angle of scattering, the level of energy dampening that occurs, and the new number of mean-free-paths until the next *collision*.

V. Parallelisation

The parallelisation strategy adopted for the algorithm has major consequences for the performance on different architectures, and there were several different approaches that could be taken in order to achieve parallel execution.

A. Over Particles

As the particle histories are independent, it is natural to parallelise over those individual particles, allowing a thread to follow a particle from birth to *census*. In Listing 1 we show a simple top-level view of the parallelisation strategy, where the `foreach` call can be threaded.

Listing 1. Pseudo-code for parallelising over particles.

```
foreach(particle) {
  loop until(reached_census) {
    calculate_time_to_events()

    if nearest(time_to_collision)
      handle_collision()
    else if nearest(time_to_facet)
      handle_facet()
    else if nearest(time_to_census)
      handle_census()
      reached_census = true
  }
}
```

There are many consequences of traversing the problem by particles, that may improve or inhibit performance on particular architectures:

- *Data is cached in registers between events* - For instance, the microscopic cross sections only need to be looked up when the energy of the particle changes, which only happens upon a *collision* event. By caching the values, consecutive *facet* encounters don't need to access the lookup tables.
- *Thread synchronisation is minimised, but a load imbalance is possible* - A single synchronisation point is present at the end of the `foreach` statement; however, the length of each particle history can vary significantly, potentially leading to a load imbalance between threads.
- *The code contains deep branches* - Threads acting upon the particles will often be divergent, as there are multiple branches that extend two to three levels deep. Divergence may be an issue for architectures such as the GPU.
- *Reading from the density mesh may exhibit locality benefits* - A particle will move from its cell to an immediate neighbour, so there is opportunity for data locality of reads from the density mesh following a single particle history.

B. Over Events

Looking at the problem from a breadth first perspective, progressing all particle histories by a single event, it is possible to parallelise over the *facet*, *collision*, and *census* events.

Listing 2. Pseudo-code for parallelising over events.

```
loop until(all_particles_reach_census) {
  foreach(particle)
    calculate_time_to_events()
    determine_next_event()

  foreach(colliding_particle)
    handle_collision()

  foreach(particle_encountering_facet)
    handle_facet()
}

foreach(particle)
  handle_census()
```

This approach exposes increased data parallelism, and has different properties to parallelising Over Particles.

- *The algorithm forms tight vectorisable loops*
- *Data can no longer be cached in registers* - Any time data is to be cached, it must be stored per particle.
- *Particles are gathered from memory* - Each kernel visits the entire list of particles checking if they are to be processed. Although the particles are contiguous, the data is gathered from memory as not all particles encounter the same event.
- *Branching is decreased* - As the events are collected together, the top level branches of the Over Particles algorithm are negated.
- *The load imbalance is removed, but synchronisation is increased* - Each of the `foreach` statements can result in a synchronisation, but the amount of work is known before the loop, allowing a static schedule.

C. Energy Deposition

In the `neutral` mini-app, we are interested in tracking the average energy deposition per cell of a computational mesh, as particles travel through it. The tally is calculated by tracking the path lengths of each particle through each cell and scaling this by an expected heating response. This creates a write dependency upon the tally mesh, which is essentially a reduction into the mesh that must be performed atomically to avoid race conditions. The energy is tallied for a particle every time it encounters a *facet* encounter, or once *census* is reached for the particle history.

VI. Analysis and Optimisation

As the mini-app was written from scratch, with no optimal design template, it was necessary to analyse and optimise every aspect of the code for each target architecture.

A. Code Analysis

There are several performance affecting steps taken by the main computational loop: (1) calculating the time until the next event and energy deposition from the path length, (2) handling the *collision* and *facet* events, and (3) updating the energy deposition tally.

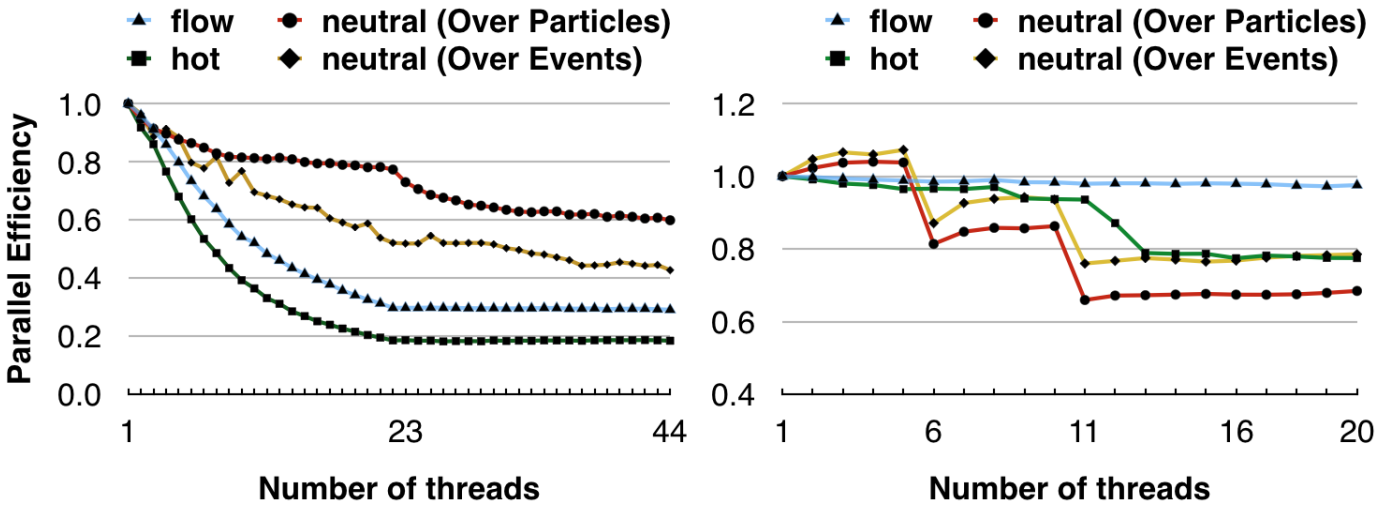

Fig. 3. The parallel efficiency of the `csp` test case as thread count is increased on the two sockets of Intel Xeon E5-2699 v4 (Broadwell) 22 core CPU, and two sockets of POWER8 10 core CPU (right).

The time calculations are essentially a Cartesian intersection check, and the energy deposition is calculated with a small closed form equation. Both amount to a limited number of FLOPS and primarily work on data present in registers.

The *collision* event contains two main branches, both comprised of little computational work, but the elastic scattering includes three calls to `sqrt`. Whenever a collision is encountered, the microscopic cross sectional tables need to be checked using the new energy. The index of the previous lookup is cached so that a fast linear search can be used to take advantage of cache locality, instead of performing a more expensive binary search at each step. This particular optimisation improved the performance of the *csp* problem by 1.3x, but might suffer issues when larger jumps in energy are observed due to physical phenomena.

The *facet* event contains many nested branches, up to four levels deep, which handle reflective boundary conditions, and updating mesh locations. Although the branching is unpredictable, the work at each branch amounts to only one or two FLOPs. Whenever a *facet* is encountered, the cached local density needs to be updated, requiring a read from the cell centered density mesh. Whenever a *collision* occurs, the energy deposition increments a register, but at the end of a *facet* encounter the value is flushed onto the tally mesh, which means every *facet* encounter results in an `atomic` read-modify-write operation.

Using the Over Particles parallelisation scheme, it is challenging to gain accurate profiling data about the individual methods, given their fine granularity. We used the *scatter* problem to calculate an average runtime of 18ns for *collision* events, and the *stream* problem to calculate an average runtime of 3ns for *facet* events. We determined with sample-based profiling on the Intel Xeon CPU that the tallying of energy deposition accounts for around 50% of the total runtime, while for the Over Events scheme it only accounts for 22% of the runtime. We believe this is due to the increased workload of other methods affecting the proportion of time spent tallying.

We essentially ignore the *census* event, as it is executed so infrequently, compared to the other events, that it does not affect the performance. It is impossible to accurately predict what the balance of events will look like for a general test problem, which is why we present results for test cases that show both extreme cases and a balanced case.

B. Thread Scaling

Given that the mini-app is new, it was an important first step to analyse the parallel scaling efficiency of the algorithms. We thought it would be insightful to consider the parallel efficiency of the algorithm compared to two applications from the `arch` project, `flow`[3] and `hot`[4]. The `flow` mini-app is a highly optimised hydrodynamics application, while `hot` is a conjugate gradient based heat conduction linear solver, both of which serve as an interesting point of comparison.

Figure 3 shows the parallel efficiency of both parallelisation schemes for `neutral` compared to `flow` as the thread count is increased on the Intel Xeon Broadwell and POWER8 CPUs. The parallel efficiency of `hot` and `flow` is limited by the fact that the algorithms are memory bandwidth bound. On the Intel Xeon CPU, we can see for both applications that the parallel efficiency drops and normalises when the second socket is consumed, if you instead interleaved the threads on NUMA nodes, the scaling drops slower, as the threads can take advantage of the second socket's memory controllers. The parallel efficiency of `neutral` is comparatively higher on a single socket. We later show that `neutral` is not bound by memory bandwidth or the available FLOPS on the CPU, and the scalability issues suggest the application is limited instead by another concern, such as instruction or memory latency. On Broadwell a key feature of the `neutral` scaling is the rapid drop in parallel efficiency that occurs as the

[3]https://github.com/uob-hpc/flow
[4]https://github.com/uob-hpc/hot

threads cross the NUMA domain onto the second socket. We would have expected that the additional cache would help performance, but this result potentially shows the latency impact of data being stored and randomly accessed across sockets. An MPI decomposition over NUMA domains could improve performance, which will make an important future research direction.

The POWER8 results for both schemes demonstrates unusual features that the authors had not encountered before. The step functions at the 6th and 11th threads are possibly caused each time the distance to cache is increased, firstly as data is forced across the on-chip interconnect between the two groups of 5 cores, and subsequently when threads are scheduled on the second socket. We can see that `flow` achieves near perfect parallel efficiency on the POWER8 CPU, as there are many memory controllers, 8 per CPU, meaning that many threads are required to saturate the memory bandwidth.

While the parallel efficiency of the Over Events scheme on the Intel Xeon CPU appears to contain timing errors, where the line is not smooth, the results are precisely reproducible. We have not yet been able to understand why the small peaks are seen at unexpected intervals.

C. Thread Scheduling

The independent nature of the particle histories in the application means that each particle will execute a different number of instructions. Profiling with VTune suggested that there might be a load imbalance, and we had originally hypothesised that the varying lengths of particle histories could lead to certain threads executing significantly more work than other threads. In order to test this, we experimented with the `schedule` clause in OpenMP across the test processors.

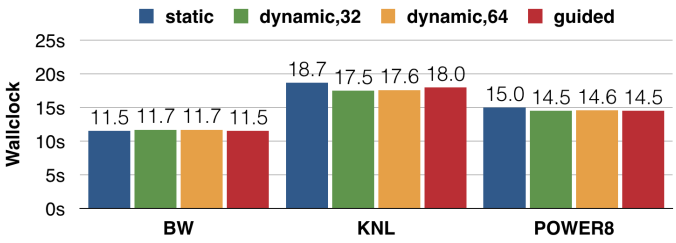

Fig. 4. The *csp* problem run on the Intel Xeon, Intel Xeon Phi, and POWER8 CPUs, respectively, with different OpenMP `parallel for` scheduling options.

Figure 4 shows different scheduling strategies for the *csp* problem, which exhibited the greatest load imbalance of the three test cases. The scheduling strategies at most improved performance by 1.07x on the KNL, suggesting the load imbalance is smaller than we had expected for our test problems. In the case of production test problems it may be that this issue is more pronounced.

D. Data Structure

An important factor in the performance of the application is the data structure used to describe particles. The most intuitive approach was to make it such that a single particle is represented by a data structure that contains the particle's position in space, direction, energy, and location on the mesh. There are also options for compacting the data structure, for instance storing the energy alongside direction in a velocity vector, or calculating the mesh cell on the fly.

When we port the code to work on the GPU, we only consider the Structure of Arrays (SoA) layout, given that this assists with coalescing memory access. In the case of CPU and KNL, we wanted to explore the performance difference between SoA and Array of Structures (AoS).

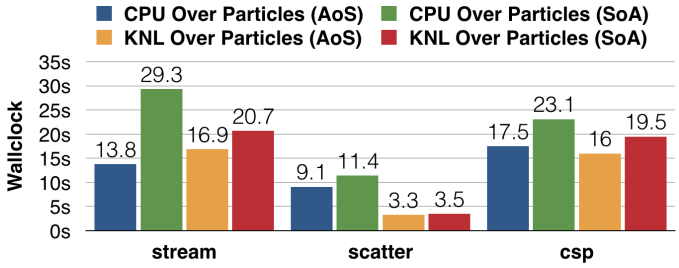

Fig. 5. SoA vs. AoS on a single socket of Intel Xeon E5-2699 v4 (Broadwell) 22 core CPU, and an Intel Xeon Phi 7210 (KNL) CPU with 256 threads.

The results in Figure 5 clearly demonstrate that, on the CPU, the SoA implementations perform worse than AoS for all test cases. We expect that the disparity in the results is caused by the improved cache utilisation for AoS with the Over Particles parallelisation scheme. Each particle, as encountered, can be loaded from cache into registers with little redundant memory access, and will be worked with for the entire particle history. In the SoA scheme, each thread loads a cache line for each particle field, and only uses a single item, which exacerbates the memory access and latency issues.

E. Hyperthreading

The results in Figure 6 show that `neutral` benefits significantly from hyperthreading. On the Intel Xeon CPU we observed as much as a 1.37x speedup when running an OpenMP thread per logical core compared to one thread per physical core. On the KNL the *csp* test case speeds up by 2.16x, while the same test problem on the POWER8 improves by 6.2x when running all 8 Simultaneous Multi-Threads (SMT).

For comparison, the `flow` application saw no improvement for running with hyperthreads, and a roughly 1.2x performance penalty for oversubscribing the number of threads to hyperthreads on the Broadwell CPU.

On Broadwell, we encountered a minor performance improvement for oversubscribing threads beyond the number of logical cores. While this would not be a useful optimisation in itself, it suggests that the context switching between threads was faster than waiting on the thread's current operation. It is not clear if there are other factors influencing the performance; however, we hypothesise that this is exposing the severe memory latency issues encountered by the application.

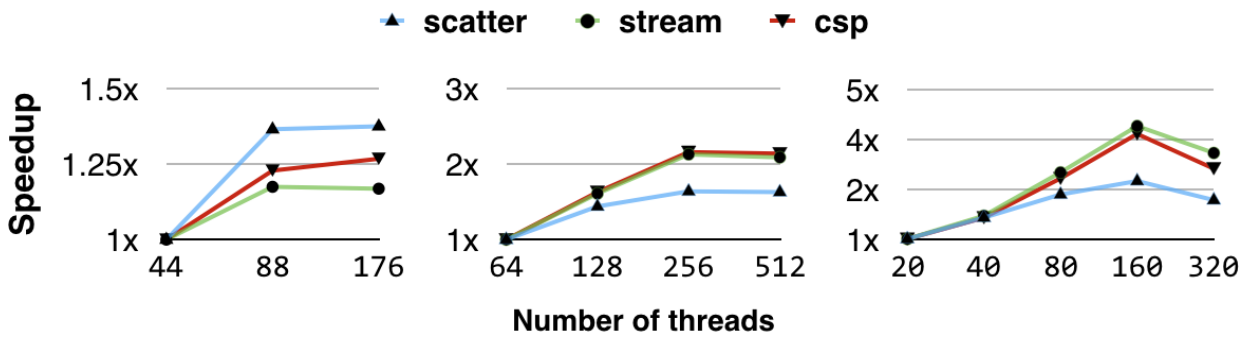

Fig. 6. Increasing the number of threads on Intel Xeon Broadwell (left), Intel Xeon Phi Knights Landing (middle), and POWER8 (right).

F. Tally Privatisation

The tally in the `neutral` mini-app is an energy deposition across the mesh. Our implementation requires an atomic operation to ensure thread safety; however, this operation is potentially expensive, as discussed in Section VI-A. One possible optimisation was to privatise the tally mesh per thread, which would completely avoid the need for the atomic operation.

A consequence of privatising the tally mesh is that the capacity of the tally data increases with the number of threads. On a KNL, where 256 threads are optimal, the total memory footprint of the *csp* test problem increases from 0.3GB to 31GB, which exceeds the maximum capacity of MCDRAM. In practice, the DRAM is faster for this application; however, this solution may encounter capacity issues with larger meshes, or on future architecture.

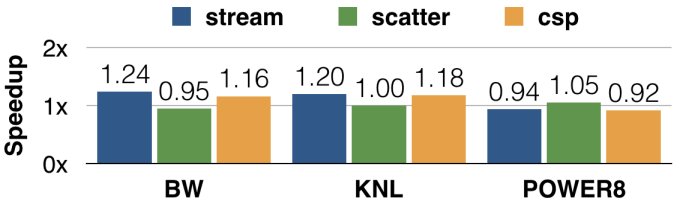

Fig. 7. Demonstrating the speedup of the mesh privatisation technique for the energy deposition tallies, across all CPU architectures and test problems.

The results in Figure 7 demonstrate that the optimisation was possible, and in some cases lead to an improved runtime for the application. On the Broadwell and KNL, a 1.16x and 1.18x speedup were seen for the *csp* problem. While this is a reasonable improvement, the extent of the atomic overhead suggested that the optimisation would achieve a more significant increase in performance. We hypothesise that the increased memory footprint caused negative cache effects, offsetting the benefit of removing the atomic operation.

A limitation of the results is that the energy deposition tally is compressed at the end of the solve, to perform validation. For a real-world use case, the application would likely be collecting tallies to update the source terms of another application, and the energy deposition would need to be merged from all threads at every timestep. This was found to make the solve significantly slower than when using atomic operations, for all test architectures.

G. Vectorisation

Successful vectorisation is often essential for good performance on the Intel Xeon and Intel Xeon Phi architectures. The Over Events scheme provided obvious vectorisation opportunities; however, it was not clear how the Over Particles loop could be successfully vectorised given the complexity of the branching within the single computational loop.

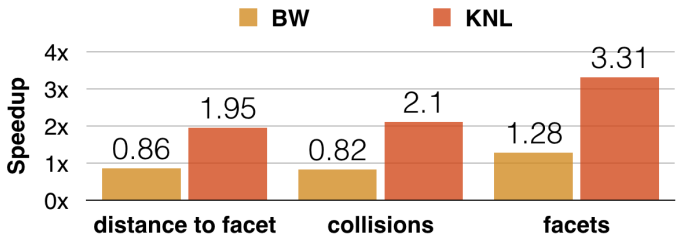

Fig. 8. Vectorisation per method of the Over Events parallelisation scheme. Speedup compared to un-vectorised code.

Initially, vectorisation of the Over Events scheme was inhibited by the atomic operation for tallying in each of the loops. We were able to work around the issue by creating a separate tally loop that handled the atomic operations after the other events had completed. The results in Figure 8 show that vectorisation only helped the *facet* events on the CPU, but that the KNL benefited significantly for all events.

The Over Particles computational loop presented a significant challenge for vectorisation, as the un-vectorisable atomic operations were pervasive, and the loop contains deep branches with varying workloads. In Section VI-F, we discussed the privatisation of the tallying mesh for the purpose of removing the atomic operation. Using this version of the code, it was possible to convince the Intel compiler version 17.2 to vectorise the entire computational loop, as the atomic had been removed from the loop; however, the performance was either not improved or worse in all cases.

H. Registers

The Over Particles scheme consists of a single large computational loop that calls multiple different methods to handle

different events. On the GPU this leads to a high number of registers per thread, which reduces the occupancy of the application, harming performance. One of the most significant optimisations we applied to this scheme involved restricting the number of registers from 102 down to 64, achieving a speedup of 1.6x for the `csp` problem.

VII. Comparing Over Particles and Over Events

Although we cannot guarantee that `neutral` is yet optimal on every architecture, we have been able to explore many avenues for optimisation. In this section we will discuss the performance of both the Over Events and Over Particles parallelisation schemes.

A. Intel Xeon Broadwell CPU

All results are for dual socket 22 core Intel Xeon E5-2699 v4 (Broadwell) CPUs @ 2.10Hz, where 88 threads are distributed with `KMP_AFFINITY=compact,granularity=fine`, compiled with ICC version 17.2. Note that the results are taking advantage of the hyperthreading provided by the CPU, as Section VI-B demonstrated a performance improvement.

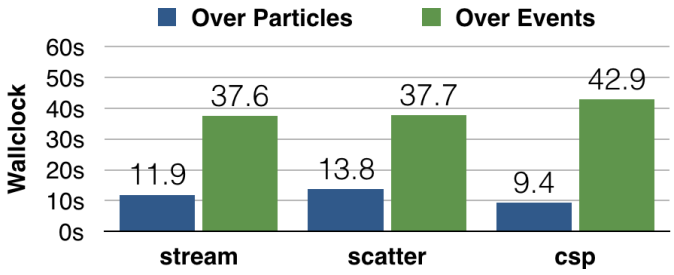

Fig. 9. Results for dual socket Intel Xeon E5-2699 v4 (Broadwell) 22 core CPUs, with 1 thread per logical core for 88 threads.

The results shown in Figure 9 unequivocally demonstrate that the performance of the Over Particles approach is optimal in all cases on the CPU. We have discovered many influential factors in this performance difference:

*1) **The atomic operations conflict less often**:* With the Over Events scheme, the `atomic` operations are batched into a single loop for every particle. The Over Particles scheme spreads the operations randomly along each particle history.

*2) **Caching occurs in registers**:* A major consequence of parallelising over particles is the increased re-use of state such as cross sections and densities. In the Over Events approach this state is cached in the particle data store and streamed from memory for each loop, greatly increasing memory traffic.

*3) **Vectorisation gave little performance benefit**:* In spite of achieving vectorisation for all of the key routines, the performance was poor due to the number of gathers and scatters required because of the indirection in each loop.

In the future we plan to extend the results of this performance analysis to consider MPI over sockets and threads bound to the hyperthreads of each CPU.

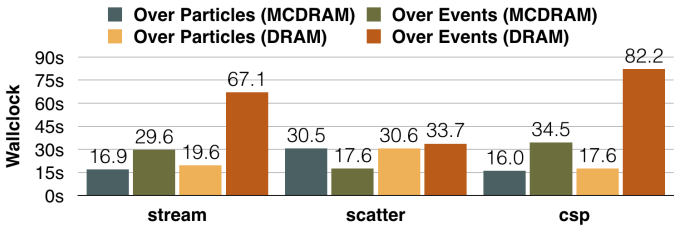

Fig. 10. Results for the different implementations on a KNL 7210.

B. Intel Xeon Phi Knights Landing

All results are for a KNL 7210, with 256 threads distributed with `KMP_AFFINITY=scatter,granularity=fine`, compiled with ICC version 17.2.

Figure 10 demonstrates the performance with all mesh and particle data either resident in MCDRAM or DRAM. For the Over Events scheme we can see that the performance is generally worse, except for in the scattering case, where the runtime is 1.73x lower than for the Over Particles scheme. In the worst case, the *csp* problem, the Over Events scheme is 2.15x slower. From profiling we understand that the improvement in performance on the KNL is due to the fact that the *collision* events are vectorised, and the memory latency issue is less important due to the few scattered loads and stores seen for highly scattering test problems. We found that for the other problems, the application spends a significant proportion of time waiting for memory to come into L2, and waiting on the completion of atomic operations.

It can be noted that the shift between DRAM and MCDRAM affects the Over Events scheme significantly more than the Over Particles scheme, with a 2.38x speedup for the *csp* problem. In spite of the poor wallclock time, this is an example of the Over Events parallelisation scheme taking advantage of the hardware more successfully. The difference is not the greatest you would expect, however, as a memory bandwidth bound problem like `flow` can observe a 5.0x increase in performance for using MCDRAM. In the *scatter* case the Over Particles approach is slightly faster when accessing DRAM, which appears to suggest that the increased MCDRAM latency is degrading performance.

C. POWER8 CPU

The results shown in Figure 11 are for dual socketed 10 core POWER8 CPUs, with 160 threads (8 SMTS), distributed with `OMP_PROC_BIND=true`, and compiled with the XL compilers version 13.1.5.

As with the Intel Xeon, and Intel Xeon Phi, the results of the Over Particles approach are significantly faster than for the Over Events approach. The difference is slightly less on the POWER8 than the Intel Xeon Broadwell, which observe a 3.75x and 4.56x respective improvement in performance for the Over Particles scheme compared to the Over Events scheme with the *csp* problem. As the performance of the POWER8 is worse than the Intel Xeon for both schemes, there may be an underlying conflict with the architecture.

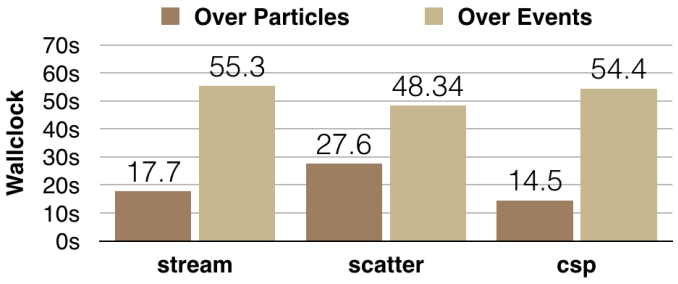

Fig. 11. Results for 160 threads of POWER8 CPU.

D. K20X GPU

In Figure 12 we include results for an NVIDIA K20X GPU, with thread block sizes of 128 threads, compiled with CUDA 8.0, as a point of comparison for the P100 GPU.

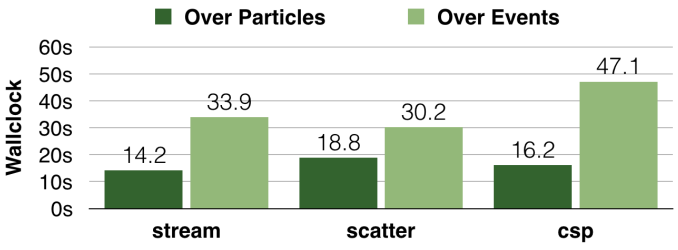

Fig. 12. Results for NVIDIA K20X GPU.

One of the major concerns we wish to address with this research is the capability of the parallelisation schemes to scale with future hardware generations, especially given that they are not bound by the architectural limits on FLOPS or memory bandwidth. The Over Particles approach achieved 35GB/s memory bandwidth for the core computational kernel, which is roughly 20% of the achievable memory bandwidth for the device. The memory bandwidth cannot be saturated by the application due to the fact that the majority of the memory access patterns are random.

For the Over Events scheme we expected that, even though the net performance was worse, we would observe high utilisation of the memory bandwidth of the device. The cached data was laid out contiguously, such that there would be a high volume of streaming traffic. Through profiling with `nvprof` we found that the approach achieved around 90GB/s or 50% of achievable memory bandwidth.

E. P100 GPU

The results in Figure 13 are for an NVIDIA P100 GPU, with 128 wide thread blocks, compiled with CUDA 8.0 for architecture version 6.0. The performance difference is again significant between the Over Particles and Over Events scheme, with the Over Particles scheme achieving 3.64x faster execution time for the *csp* problem. This suggests that, in spite of the Over Events scheme appearing to take better advantage of the hardware resources, the method is not scaling with the improvements to the architecture. The Over Particles scheme, on the other hand, has improved by 4.5x on the new architectural generation.

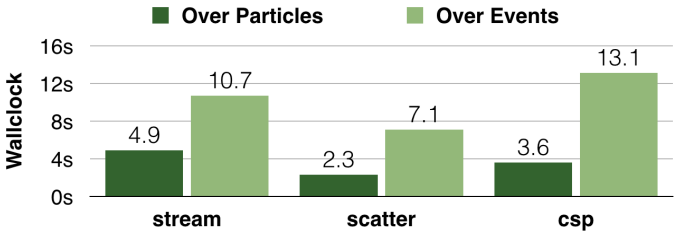

Fig. 13. Results for NVIDIA P100 GPU.

We observed that compiling for the CUDA architecture version 6.0 used 79 registers for the particle loop, while version 3.5 needed 102 registers. Although restricting the maximum number of registers greatly improves the performance on the K20X, we did not see the same improvements on the P100, in spite of the profiler recommending this action. Restricting the maximum number of registers to 64 improved the occupancy of the main kernel from 0.38 up to 0.49, but the wallclock increased by 1.07x, showing that the P100 does not require as high occupancy as previous architecture generations for maximum performance.

The achieved memory bandwidth on the P100 is 125GB/s, which equates to 25% of achievable memory bandwidth on the device. The P100 has more, smaller streaming multiprocessors (SM) than the K20X, allowing for a net increase in the number of active warps. As a consequence, it is possible for additional concurrent memory requests, hiding some of the memory latency and increasing the bandwidth utilisation.

The NVIDIA Visual Profiler suggested that 87% of the computational kernel's time was spent waiting for a memory dependency, with the remaining time spent waiting on execution dependencies, confirming our suspicion of memory latency issues. The profiler showed that the random access to the density mesh was inefficient, contributing significantly to the latency issues. It also showed that the branches of the *facet* event were divergent, causing warp inefficiencies; however, given the low grind time of the facet events, and the limited work in each branch, it does not appear to cause a significant performance issue.

VIII. Final Performance Results

At this stage we can depart from the Over Events scheme, and focus on the performance differences between the architectures for the Over Particles scheme, which was superior in almost every case, and demonstrated better scaling across hardware generations.

The KNL results did not match our expectations, and were beaten in almost all cases by the other architectures. Investigation into the root cause of this issue is still ongoing, but it may relate to the smaller caches on the KNL, or the more complicated routing in the processor affecting the latency of the application. The POWER8 achieves similar performance to the KNL on the *csp* problem, while the Intel Xeon Broadwell was 1.34x faster than the POWER8.

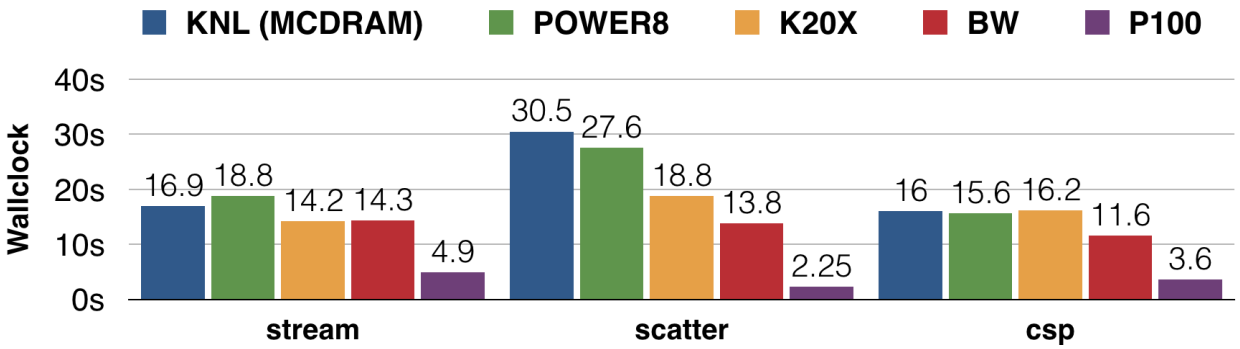

Fig. 14. Results for all tested devices with the Over Particles parallelisation scheme.

The K20X results were improved with some minor optimisations, but were actually the slowest by a small margin for the *csp* problem. Our expectation was that the divergent code would not be suited at all to the SIMD architecture of the GPU, which originally lead us to consider the Over Events parallelisation scheme, and we were surprised the results of the Over Particles scheme were not worse.

Figure 14 shows that the P100 is by far the fastest processor for `neutral` as it is currently written. The *csp* problem observes a 3.2x speedup compared to the dual socket Broadwell, in spite of the algorithm not fully exploiting of the high memory bandwidth or FLOPS. Compared to its predecessor, the K20X, the P100 has increased performance by 4.5x, which is an outstanding improvement across generations.

A. Architectural Advantages in the P100

Given that the P100 performed so well for this particular algorithm, we wanted to consider the root cause of the performance gap so that we can reason about the future architectural requirements of the algorithm.

The Broadwell CPU is limited to a small finite number of memory transactions per core, whereas the mechanisms with which threads are scheduled on a K20X means that more simultaneous in-flight memory requests are possible. Further to this, the P100 allows even more in-flight memory requests than the K20X, which will improve the random memory access issues that the algorithm suffers from.

A notable difference between the P100 and K20X for this particular application is that we had to simulate the atomic operation on the K20X. On the P100, a hardware supported double precision `atomicAdd` instruction is now available. We experimentally determined that the improvement in performance provided by this intrinsic was 1.20x.

We hypothesise that the intrinsic double precision atomics, high memory bandwidth and utilisation, and latency hiding characteristics of the P100 all contribute to the performance advantage over the Intel Xeon Broadwell and K20X.

IX. Future Work

Although the optimisations considered in this research are extensive, there still remain other potential opportunities that we would like to explore. On NUMA architectures it is possible that MPI decomposition will help to improve some of the NUMA issues that we experienced with our OpenMP implementation, and we plan to introduce a high performance, load balanced, MPI parallelisation into the application. Further to this, there are a number of other physical processes that can be modeled, which may or may not affect the performance of the mini-app. We are keen that fission, complex geometries, and multi-material meshes etc. are explored with the mini-app.

X. Related Work

Much research has focussed upon the development of robust Monte Carlo applications, for instance the Milagro code from Los Alomos National Laboratory [17], and the Mercury code from Lawrence Livermore National Laboratories [9]. Romano et al. [18] from MIT developed a new Monte Carlo particle transport application, OpenMC, and demonstrated ideal weak scaling above 100,000 processors. Siegel et al. [19] later considered the threading of the OpenMC application.

Gentile et al. [12], discuss the algorithmic and performance characteristics of the Monte Carlo and Implicit Monte Carlo methods. Horelik et al. [20], discuss the capacity issues faced when calculating tallies for full-core reactor benchmarks.

Mini-apps have been used extensively for performance optimisation studies, including porting applications onto diverse hardware, with our group has contributing heavily to this space [4], [6], [21]. Recently, we discussed the `arch` project, which the `neutral` mini-app belongs to, including a description of the design of each mini-app and potential research opportunities presented by the suite [22]. Other important studies include the optimisation of the CloverLeaf hydrodynamics application [23], [24], and the efforts of Bird et al. [25] to develop a portable particle in cell code.

XI. Conclusion

The `neutral` mini-app has shown the difficulties associated with mapping Monte Carlo particle transport onto existing architectures. We have been able to characterise the algorithm as memory latency bound, and all attempts to circumvent this characteristic resulted in worse performance overall. In contrast, modern architectures are focussing on FLOPS, and more recently memory bandwidth, with memory latency performance generally worsening.

This research has demonstrated that the Over Particles approach was more than 2x faster than the Over Events approach for our test cases and tested hardware, primarily due to the reduced synchronisation and memory access requirements. The modern HPC CPUs were quite close in performance to the K20X, except the Intel Xeon Broadwell, which was more than 1.3x faster than the other CPUs. Moving from the K20X to the P100 saw a 4.5x performance improvement for the Over Particles approach, suggesting that future generations of hardware can continue to improve the performance of this algorithm, even though memory bandwidth and FLOPS are not fully saturated.

The random memory access pattern that the algorithm exhibits is the primary bottleneck in the application, making it highly sensitive to memory latency. Our results have demonstrated that the latency tolerant nature of GPUs has provided significantly higher performance than the other processors. The P100 GPU was more than 3x faster than the CPUs, even though the algorithm appears incompatible with the architecture. We are open to the possibility that there are other optimisations that could improve the performance of the application on the tested architecture, and we plan to continue the research independently and alongside the vendors to explore this.

Acknowledgements

The authors would like to thank EPSRC for funding this research. We also extend thanks to the Intel Parallel Computing Centre at the University of Bristol, for providing access to the Zoo testbed. Further, we thank the GW4 consortium for access to the Isambard supercomputer, which provided Intel Xeon CPUs, KNLs, and NVIDIA P100 GPUs.

References

[1] Los Alomos National Laboratory, "Trinity: Advanced Technology System," 2017. [Online]. Available: www.lanl.gov/projects/trinity

[2] Lawrence Livermore National Laboratory, "CORAL/Sierra," 2017. [Online]. Available: https://asc.llnl.gov/coral-info

[3] Oak Ridge National Laboratory, "Summit: Oak Ridge National Laboratory's next High Performance Supercomputer," 2017. [Online]. Available: https://www.olcf.ornl.gov/summit/

[4] M. Martineau, S. McIntosh-Smith, M. Boulton, and W. Gaudin, "An Evaluation of Emerging Many-Core Parallel Programming Models," in *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM'16, 2016.

[5] McIntosh-Smith, S. and Price, J. and Sessions, R.B. and Ibarra, A.A., "High Performance in Silico Virtual Drug Screening On Many-Core Processors," *International Journal of High Performance Computing Applications*, 2014.

[6] T. Deakin, S. McIntosh-Smith, M. Martineau, and W. Gaudin, "An improved parallelism scheme for deterministic discrete ordinates transport," *International Journal of High Performance Computing Applications*, 2016.

[7] E. Lewis and W. Miller, *Computational methods of neutron transport*. John Wiley and Sons, Inc.,New York, NY, Jan 1984.

[8] P. Andreo, "Monte Carlo techniques in medical radiation physics," *Physics in medicine and biology*, vol. 36, no. 7, p. 861, 1991.

[9] R. Procassini, M. OBrien, and J. Taylor, "Load balancing of parallel Monte Carlo transport calculations," *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, Palais des Papes, Avignon, Fra*, 2005.

[10] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis *et al.*, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006.

[11] T. P. Wilcox Jr, "COG: A particle transport code designed to solve the Boltzmann equation for deep-penetration (shielding) problems: Volume 1: User's Manual," Lawrence Livermore National Lab., CA (USA), Tech. Rep., 1989.

[12] N. Gentile, "Monte Carlo Particle Transport: Algorithm and Performance Overview." Livermore, CA: Lawrence Livermore, 2005.

[13] P. K. Romano, "Parallel algorithms for Monte Carlo particle transport simulation on exascale computing architectures," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[14] A. Siegel, K. Smith, K. Felker, P. Romano, B. Forget, and P. Beckman, "Improved cache performance in Monte Carlo transport calculations using energy banding," *Computer Physics Communications*, vol. 185, no. 4, pp. 1195–1199, 2014.

[15] I. Lux and L. Koblinger, *Monte Carlo particle transport methods: neutron and photon calculations*. Citeseer, 1991, vol. 102.

[16] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, "Parallel random numbers: as easy as 1, 2, 3," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–12.

[17] T. J. Urbatsch and T. M. Evans, "Milagro Version 2 An Implicit Monte Carlo Code for Thermal Radiative Transfer: Capabilities, Development, and Usage," Los Alamos National Laboratory (LANL), Los Alamos, NM, Tech. Rep., 2006.

[18] P. K. Romano and B. Forget, "The OpenMC Monte Carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274 – 281, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306454912003283

[19] A. R. Siegel, K. Smith, P. K. Romano, B. Forget, and K. G. Felker, "Multi-core performance studies of a monte carlo neutron transport code," *The International Journal of High Performance Computing Applications*, vol. 28, no. 1, pp. 87–96, 2014. [Online]. Available: http://dx.doi.org/10.1177/1094342013492179

[20] N. Horelik, B. Forget, K. Smith, and A. Siegel, "Domain decomposition and terabyte tallies with the OpenMC Monte Carlo neutron transport code," Tech. Rep., 2015.

[21] S. McIntosh-Smith, M. Boulton, D. Curran, and J. Price, "On the Performance Portability of Structured Grid Codes on Many-Core Computer Architectures," in *Supercomputing*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8488, pp. 53–75.

[22] M. Martineau and S. McIntosh-Smith, "The arch Project: Physics Mini-apps for Algorithmic Exploration and Evaluating Programming Environments on HPC Architectures," in *The International Workshop on Representative Applications (WRAp)*, 2017.

[23] A. Mallinson, D. Beckingsale, W. Gaudin *et al.*, "Towards Portable Performance for Explicit Hydrodynamics Codes," 2013.

[24] W. Gaudin, A. Mallinson, O. Perks, J. Herdman, D. Beckingsale, J. Levesque, and S. Jarvis, "Optimising Hydrodynamics applications for the Cray XC30 with the application tool suite," *The Cray User Group*, pp. 4–8, 2014.

[25] R. F. Bird, S. J. Pennycook, S. A. Wright, and S. A. Jarvis, "Towards a Portable and Future-Proof Particle-in-Cell Plasma Physics Code," *Proceedings of 1st International Workshop on OpenCL (IWOCL 13)*, 2013.